Statistical Techniques in Robotics (16-831, S20) Lecture #04 (Monday, January 13)

Online Techniques for Support Vector Machines (SVMs)

Lecturer: Kris Kitani *Scribes: Jeet Kanjani, Shubham Gupta*

1 Review

In the last lectures, we studied the concepts of Online Gradient Descent (OGD) and Stochastic Gradient Descent (SGD) and how it is applied to solve the online learning problems. In the next sections, we first recap the SGD algorithm, build the necessary mathematical fundamentals and finally use them to solve the support vector machines. We finally draw comparisons between the perceptron algorithm and support vector machines.

In supervised learning, we have a separate training and test set where the algorithm learns using the examples in the training set and validates the performance on the test set. Online learning however has a single dataset that comes in the form of the data stream. We predict on this data and simultaneously use it for training as well. This can be viewed as an interleaving of training and test set in the supervised setting. In essence, online learning interweaves the training and testing steps.

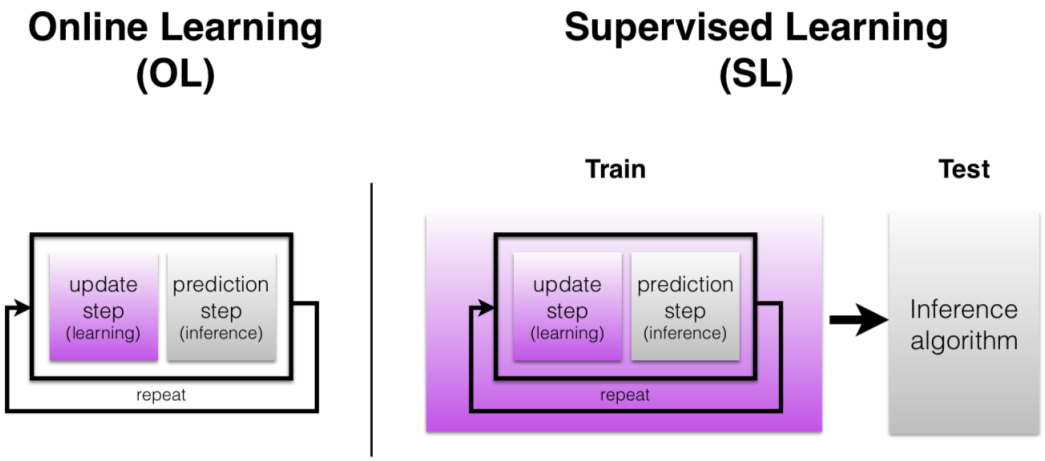

Figure 1: Online Learning v/s Supervised Learning

1.1 Online (Projected Sub-) Gradient descent

Recall, the regularization and the loss function as:

$$\begin{aligned} \psi(w) &= \frac{1}{2\eta}||w||^2 \\ f(w) &= \langle w, \theta \rangle \end{aligned} \tag{1}$$

Algorithm 1 Online SUB-GRADIENT DESCENT

1: **for** t=1,2,3...T **do**
2: $\theta^{(t+1)} = \theta^{(t)} - \eta z^{(t)}$ // with some assumption on the magnitude of the sub gradient
3: $w^{(t+1)} = -\eta\theta^{(t+1)}$ // Mirror projection
4: **end for**

Algorithm 2 Online PROJECTED SUB-GRADIENT DESCENT

1: **for** t=1,2,3...T **do**
2: $\theta^{(t+1)} = \theta^{(t)} - \eta z^{(t)}$ // Dual paramter update
3: $w^{(t+1)} = \prod_{\theta \to S} -\eta\theta^{(t+1)}$ // Mirror projection
4: **end for**

2 Summary

2.1 Hyperplane

A hyperplane is a plane that divides the n-dimensional data points into two components. It is called a line in 2D and a plane in the case of 3D.

2.1.1 Distance of line from the origin

For an arbitrary line wx + b, the distance of the origin to the line is given by:

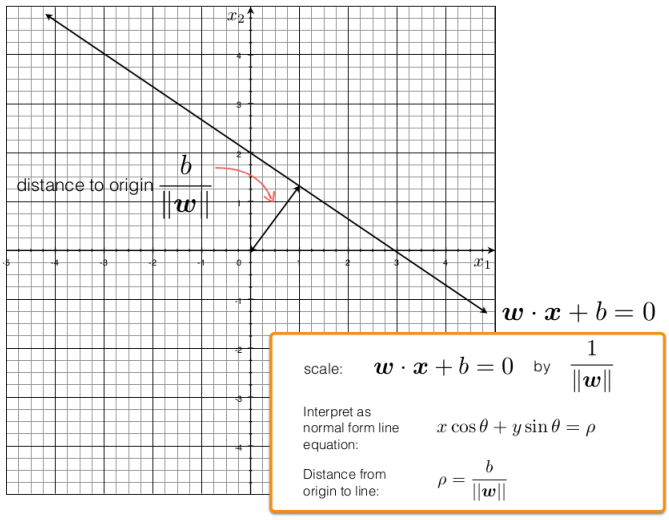

Figure 2: Distance of line from the origin

Distance of a hyperplane y = $w \cdot x_i$ + b where w $\in R^N$ from the origin is also given by $\frac{b}{||w||}$.

2.1.2 Margin

The distance between two parallel hyperplanes generated by shifting the planes by one in and opposite to the direction of w is given by:

$$\begin{aligned} M &= \frac{(1-b)}{(|W|)} - \frac{(-1-b)}{(|W|)} \\ M &= \frac{2}{|W|} \end{aligned} \tag{2}$$

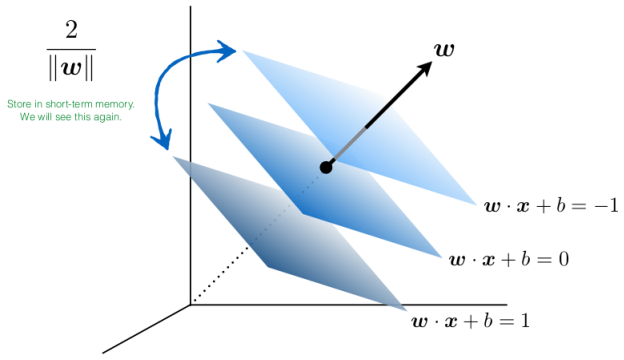

Figure 3: Margin of the hyperplane

The margin of the hyperplane is the minimum distance between the hyperplane and the closest data point.

2.2 Optimal margin

An optimal hyperplane is the one which maximizes this margin. In canonical form it is given by[1]:

$$\max_{w} \left(\frac{2}{||w||} \right)$$

$$\text{subject to, } w.x_i + b \begin{Bmatrix} \geq +1 \text{ if } y_i = +1 \\ \leq -1 \text{ if } y_i = -1 \end{Bmatrix} \text{ for i = 1,...,N} \tag{3}$$

Since the l2 optimizations are much more stable than the l1 optimization, the above equation could be written as[3]:

$$\min_{w} \left(||w||^2 \right)$$

$$\text{subject to, } y_i(w.x_i + b) \geq 1 \quad \text{for i = 1,...,N} \tag{4}$$

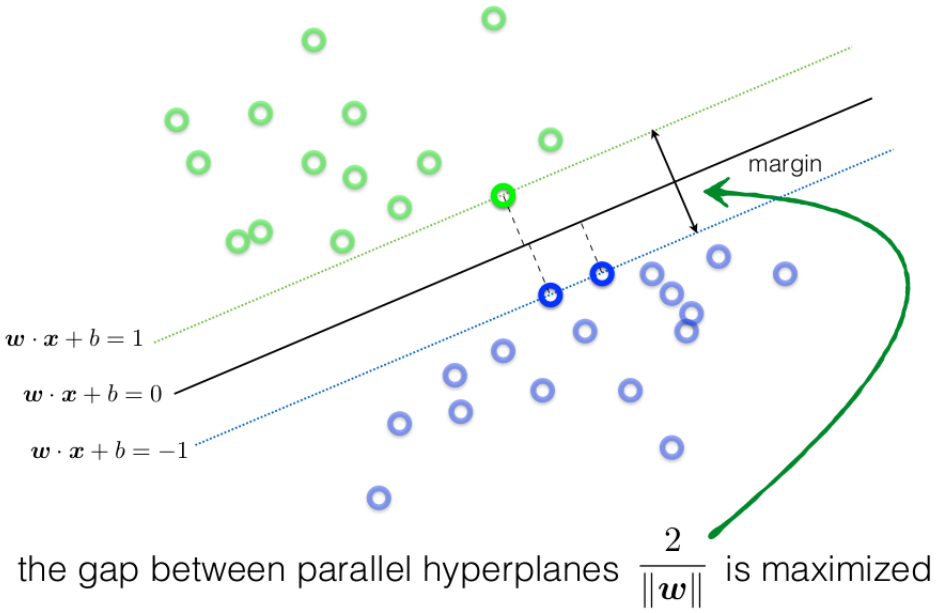

Figure 4: Max Margin of the hyperplane

This notational trick of $y_i(w.x_i + b)$ only works when doing binary classification. The loss function is changed to a quadratic such that we can make a connection to the online convex optimization. This is also called the **primal** form of a linear SVM. The optimization is a convex quadratic programming problem.

2.3 Soft Margin SVM

When the data is not linearly separable, the soft margin allows the SVM to make a certain number of mistakes and keep the margin as wide as possible so that other points can still be classified correctly[2]. To achieve this, some relaxation is put on the hard constraint $y_i(w.x_i + b)$ by introducing slack variable ξ:

$$\min_{w,\xi} \left(||w||^2 + C \sum_i \xi_i \right) \tag{5}$$
$$\text{subject to, } y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \text{for i = 1,...,N, } \xi_i > 0$$

here C is a regularization hyperparameter that decides the trade-off between maximizing the margni and minimizing the mistakes. When C is small, classification mistakes are given less importance and focus is more on maximizing the margin. Whereas when C is large, emphasis is on avoiding misclassification at the expense of keeping the margin small. Since, we have a linear objective with a quadratic regularizer, we can use stochastic gradient descent to solve this. On merging the linear constraints into the objective function, we get:

$$\min_{w} \left(\frac{\lambda}{2} ||w||^2 + \frac{1}{M} \sum_{m=1}^{M} (1 - y_m w^T x_m) \right) \tag{6}$$

However, in our objective function, we are interested in the points which are incorrectly classified and the weakly correct ones (ignoring the very correct ones).

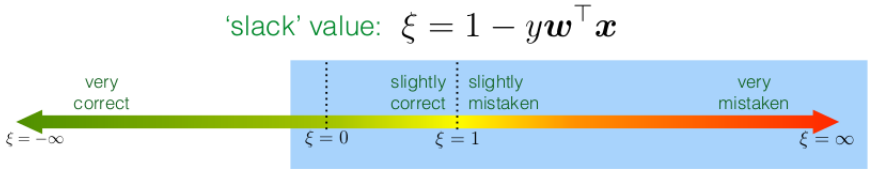

Figure 5: points in slightly correct, slightly mistaken and very mistaken are the ones objective function should be optimized on.

The way to achieve this is Hinge Loss:

$$\min_{w} \left(\frac{\lambda}{2} ||w||^2 + \frac{1}{M} \sum_{m=1}^{M} \max\{0, 1 - y_m w^T x_m\} \right) \tag{7}$$

Now that we have a convex but not differentiable function (due to the hinge), we can use sub-gradient descent to optimize this.

2.4 Sub-gradients

The gradient descent algorithm discussed above requires the loss function to be differentiable. To use it for this class of non-differentiable convex functions, we have to do certain modifications. We achieve this by introducing the concept of subgradients. Subgradient refers to the possible set of gradients for a non-differentiable convex function.

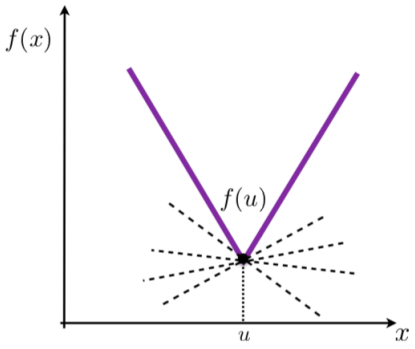

Figure 6: Multiple gradients (sub-gradients) at $x = u$ for $f(x)$

Consider the function given in the above image. We can consider any of the possible gradients at point $x = u$ for the function $f(u)$. We call this set of possible gradients at point $x = u$ as sub-gradients for $f(u)$.

2.5 Online Sub-gradient Descent

Online sub-gradient descent is an algorithm used to assign gradient to a non-differentiable convex function. The algorithm uses any one of all the available sub-gradients for the non-differentiable points. Thus, we can use sub-gradient descent for soft-margin SVM optimization.
Recall soft margin SVM loss is given by

$$\min_{w} \left(\frac{\lambda}{2}||w||^2 + \frac{1}{M} \sum_{m=1}^{M} \max\{0, 1 - y_m w^T x_m\} \right)$$

Here, the second term ($\frac{1}{M} \sum_{m=1}^{M} \max\{0, 1 - y_m w^T x_m\}$) is hinge loss (non-differentiable).

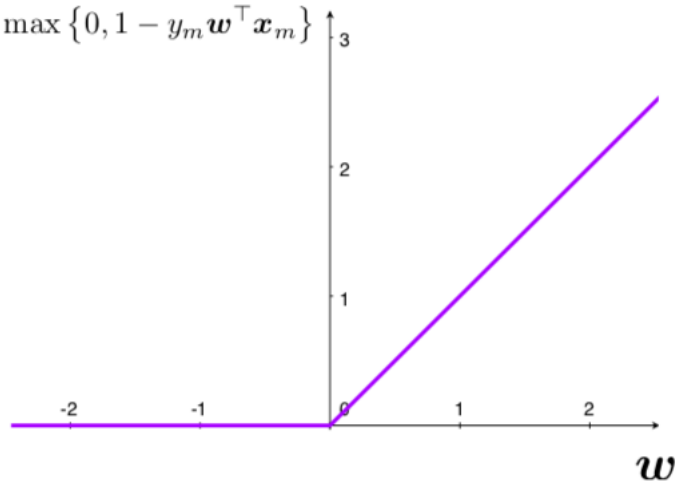

Figure 7: Hinge Loss

We consider 2 possible sub-gradients to optimize this loss function.

$$z_m = \begin{cases} 0 & \text{if } y_m w^T x_m \geq 1, \\ -y_m x_m & \text{otherwise} \end{cases}$$

Case 1: Note that when $y_m w^T x_m$ is greater than 1, it means that the point is correctly classified by the algorithm and far from the margin. We consider the hinge loss gradient to be zero in this case.
Case 2 When $y_m w^T x_m$ is less than 1, it means that we have either incorrect classification, or the point lies within the margin. We consider the gradient of loss to be $-y_m x_m$ in this case.

2.6 Soft Margin SVM Optimization using Online Sub-gradient Descent

Using the gradient derived for hinge loss in the previous section, we can write an update rule for the weights. Below is the pseudo code for the algorithm **Pseudo Code:**

Algorithm 3 SoftSVM(λ)

1: $\theta^{(1)} \leftarrow 0 \in R^N$ // Written in the **OMD** format
2: **for** t=1,2,3...T **do**
3: $x_d, y_d \sim D$ // Receive sample from environment
4: $\theta^{(t)} = \theta^{(t-1)} + y_d x_d . 1[y_d(w^{(t)}.x_d) < 1]$ // Dual parameter update
5: $w^{(t+1)} \leftarrow \frac{1}{\lambda(t+1)}\theta^{(t)}$ // Mirror Projection
6: **end for**

2.7 Comparison with Perceptron

Recall that the perceptron algorithm in an online setting is given as follows
Pseudo Code:

Algorithm 4 Perceptron

1: $w^{(1)} \leftarrow 0$ // Written in the **OMD** format
2: **for** t=1,2,3...T **do**
3: RECEIVE $x^{(t)} \in R^N$ // Receive input from environment
4: $\hat{y}^t = sign(w^{(t)} \cdot x^{(t)})$
5: RECEIVE $y^{(t)} \in \{1, -1\}$ // Get the Target
6: $w^{(t+1)} = w^{(t)} + y^t x^{(t)} . 1[y^{(t)} \neq \hat{y}^{(t)}]$ // Weight Update
7: **end for**

Putting these side by side for comparison, we can see that only 2 lines (highlighted) differ in both the algorithms.

1: **function** SoftSVM(λ)
2: $\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0} \in \mathbb{R}^N$
3: **for** $t = 1, \ldots, T$ **do**
4: $y_d, \boldsymbol{x}_d \sim \mathbf{D}$
5: $\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} + y_d \boldsymbol{x}_d \cdot \mathbb{1}\left[y_d(\boldsymbol{w}^{(t)} \cdot \boldsymbol{x}_d) < 1\right]$
6: $\boldsymbol{w}^{(t+1)} \leftarrow \frac{1}{\lambda(t+1)}\boldsymbol{\theta}^{(t)}$

1: **function** Perceptron Algorithm
2: $\boldsymbol{w}^{(1)} \leftarrow \mathbf{0}$
3: **for** $t = 1, \ldots, T$ **do**
4: Receive$(\boldsymbol{x}^{(t)}, y^{(t)})$
5: $\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t)} + y^{(t)}\boldsymbol{x}^{(t)} \cdot \mathbb{1}[y^{(t)}\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}^{(t)} \rangle < 0]$
6: $\boldsymbol{w}^{(t+1)} = \boldsymbol{\theta}^{(t)}$

Figure 8: Comparison between SoftSVM and Perceptron Algorithm

We can observe that both SVM and perceptron are almost similar - both have piece-wise linear loss with quadratic regularization. Both use similar dual parameter updates and share the same mirror function. But there are 2 key differences.

1. SVM uses a soft margin of 1 in line 4 whereas, perceptron doesn't have a concept of margin.

All perceptron cares about whether all labels are correctly classified or not.

2. The weight update rule has a multiplication factor in the SVM whereas for perceptron there is no such factor.

References

[1] C Bishop, Pattern Recognition and Machine Learning, 2005

[2] R Mishra, SVM soft margin formulation and kernel trick
https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

[3] MlMathIO Math behind support vector machine
https://medium.com/@ankitnitjsr13/math-behind-support-vector-machine-svm-5e7376d0ee4d